Software Requirements Specification

Caffeine Coders | 9 July 2024

Fueling innovation, one cup at a time!

Document Table of Contents:

Glossary 3
Software Architecture 4
Information Domain 6
- Server 6
- Database 7
- GUI 8

Functional Domain 9
- Server 9
- Database 10
- GUI 10

Interfaces 12
- Server 12
- Database 12
- GUI 13

Functional Requirements 13
Non-Functional Requirements 13
Implementation Constraints 13
GUI Paper Prototype 13
Use case document 14
Supplement specification 20
GUI Prototype Document 23
Lessons Learned 24
Credits List 24

Glossary

- **Firestore database** - NoSQL cloud database, built on Google Cloud infrastructure, to store and sync data for client- and server-side development.

- **Firebase Authentication** - backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to your app. It supports authentication using passwords, phone numbers, popular federated identity providers like Google, Facebook and Twitter, and more.

- **Express Server** - A minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It facilitates the creation of web servers and API.

- **REST API** - Representational State Transfer Application Programming Interface. It's an architectural style for designing networked applications, allowing communication between a client and server using HTTP methods.

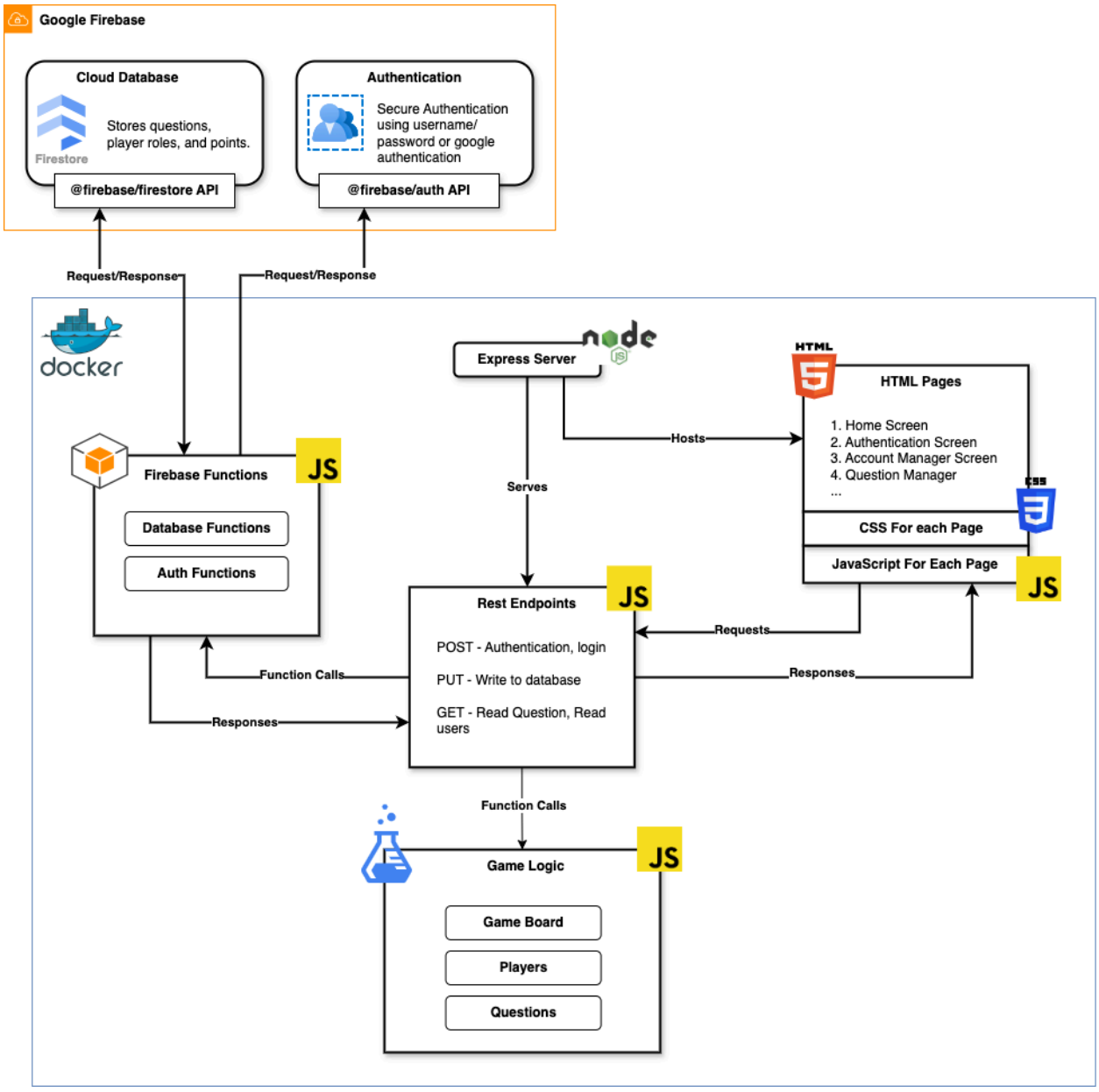
- **NoSQL** - "Not Only SQL" database, a non-relational database design that provides flexible schemas for storage and retrieval of data. It's often used for large sets of distributed data.

- **GUI** - Graphical User Interface. The visual part of a computer application or operating system through which users interact with electronic devices using visual indicators and graphical icons.

- **Game Board** - In the context of this project, it refers to the virtual playing surface where the trivia game takes place. It likely includes spaces for players to move, different categories, and possibly special tiles or spaces.

Software Architecture

This diagram depicts the architecture of the Trivial Compute system and its subsystems. The architecture follows a client-server model with cloud services integration. The central piece is the Express Server, which acts as the backbone of the application, responsible for serving the application and handling REST endpoints.



Subsystems:

1. **Cloud Services (Google Firebase):**
    - **Database:** Stores game data, user information, and questions/answers.
    - **Authentication:** Manages user authentication and authorization.
2. **Server:**
    - **Rest Endpoints:** Handles HTTP requests from the client, serving as the API for the application.
    - **Firebase Functions:** Serverless functions for backend operations that integrate with Firebase services.
    - **Game Logic:**
        - Manages game state
        - Handles player turns and movements
        - Processes question selection and answer validation
        - Updates scores and game progress
3. **GUI:**
    - **HTML pages:** Structures the content of web pages.
    - **CSS:** Styles the HTML elements for visual presentation
    - **JavaScript:**
        - Handles client-side logic and interactivity
        - Communicates with the server via API calls
        - Updates the DOM based on game state

Information Domain

This section describes what information each subsystem encapsulates.

Database

The database, implemented using Firebase Firestore, stores:

1. **Questions and Answers:**
    - Question text
    - Answer options (for multiple choice questions)
    - Correct answer(s)
    - Category
    - Difficulty level
    - Type (text, image, audio, video)
    - Usage statistics (times asked, correct answer percentage)
2. **User Profiles:**
    - User ID
    - Username
    - Email (for authentication)
    - Password hash (if using email/password authentication)
    - Game history (games played, win/loss record)
    - Achievement tracking
3. **Game Sessions:**
    - Session ID
    - Participating players
    - Current game state (board configuration, player positions, scores)
    - Game settings (number of questions, categories in play, custom rules)
    - Chat logs (if implementing a chat feature)
4. **Media Content:**
    - References to cloud storage locations for audio, video, and image content associated with questions
    - Metadata (file type, size, duration for audio/video)
5. **Categories:**
    - Category names
    - Descriptions
    - Associated color or icon references

Server

The Express server is responsible for managing the game logic, REST endpoints, and database functions. It encapsulates various components essential to the game's operation:

1. **Game Logic:**
    - Game Board:
        - Represents the logical structure of the game board in a manipulatable data structure
        - Tracks special tiles or spaces (e.g., category-specific spots, bonus spaces)
        - Manages board navigation and player movement rules
    - Players:
        - Names
        - Locations on the game board
        - Current scores
        - Turn order
        - Special abilities or power-ups (if applicable)
    - Questions:
        - Logic for selecting questions based on category and difficulty
        - Answer validation mechanisms
        - Tracking of asked questions to prevent repetition
2. **Game State Management:**
    - Current game phase (setup, in progress, ended)
    - Active player's turn
    - Dice roll results and movement calculations
    - Timer management for timed questions or turns
3. **REST API:**
    - Endpoint definitions and handlers
    - Request parsing and response formatting
    - Error handling and status codes
4. **Database Interaction:**
    - Query construction and execution
    - Data transformation between server and database formats
5. **Session Handling:**
    - Creation and management of game sessions
    - Player joining/leaving mechanisms

GUI

The GUI contains the graphical information for each page, including:

1. **Page Layouts:**
    - Structure and organization of elements for each view (home page, game board, question display, leaderboards)
    - Responsive design specifications for different screen sizes and orientations
2. **Styles:**
    - Color schemes and palettes
    - Typography (font families, sizes, weights)
    - Button and input field styles
    - Icon sets and usage guidelines
3. **Interactive Elements:**
    - Game board visualization
    - Question and answer interfaces (multiple choice, true/false, free text input)
    - Player controls (dice rolling mechanism, answer submission, turn actions)
    - Scoreboard and player status displays
    - Animations for player movements, score updates, and transitions
4. **Asset Information:**
    - Image files for board spaces, player tokens, and UI elements
    - Audio files for sound effects and background music
    - Video players for video-based questions
5. **Client-side State:**
    - Temporary storage of game state for smooth interactions
    - User input before submission to the server
    - Caching of frequently accessed data (e.g., player info, recent questions)
6. **Accessibility Features:**
    - ARIA labels and roles for screen readers
    - Keyboard navigation support
    - Color contrast information for visibility

Functional Domain

This section describes the functions of each subsystem.

Server

The game logic on the server performs the following functions:

1. **Game Management:**
    - Establish players in the game and manage turn changes: Initialize players at the start of the game, track whose turn it is, and switch turns smoothly.
    - Handle game setup and configuration: Set up the game board, initialize game settings, and manage game modes.
    - Manage game state: Track and update the overall game state, including game phases (setup, in-progress, end-game).
2. **Player Interaction:**
    - Facilitate player movement around the game board: Enable players to move their pieces around the game board based on dice rolls and game rules.
    - Handle dice rolling mechanics: Simulate dice rolls that determine how many spaces the players are allowed to move on the board.
    - Process player actions: Handle player decisions, such as choosing categories or using power-ups (if applicable).
3. **Question and Answer System:**
    - Implement question and answer mechanics: Present questions to players, validate their answers, and provide feedback based on correctness.
    - Manage question selection: Choose appropriate questions based on category, difficulty, and ensuring variety.
    - Track asked questions: Prevent repetition of questions within a game or session.
4. **Scoring and Progress:**
    - Update player scores: Adjust scores in real-time based on players' actions, correct answers, and game progress.
    - Track achievements: Monitor and award achievements or milestones reached by players.
5. **API and Communication:**
    - Handle REST API requests: Process incoming requests from the GUI, execute appropriate game logic, and send responses.
    - Manage real-time updates: Use WebSockets or similar technology to push real-time game state updates to clients.

6. **Integration with Database:**
    - Fetch and cache game content: Retrieve questions, user data, and game settings from the database efficiently.
    - Persist game state: Save ongoing game progress to allow for game resumption or analysis.

Database

The database system supports the following functions:

1. **Data Management:**
    - Create, read, update, and delete (CRUD) operations for questions, user accounts, and game sessions.
    - Support efficient querying and indexing of game data for quick retrieval.
2. **User Management:**
    - Store and retrieve user profiles, including authentication information and game history.
    - Track user statistics and achievements across multiple game sessions.
3. **Game Content Management:**
    - Organize and categorize questions by difficulty, type, and category.
    - Manage media content associated with questions (e.g., images, audio, video).
4. **Session Handling:**
    - Create and manage game session data, including current state and player information.
    - Support concurrent game sessions for multiple users.
5. **Analytics Support:**
    - Store and retrieve game analytics data for performance analysis and game improvement.

GUI

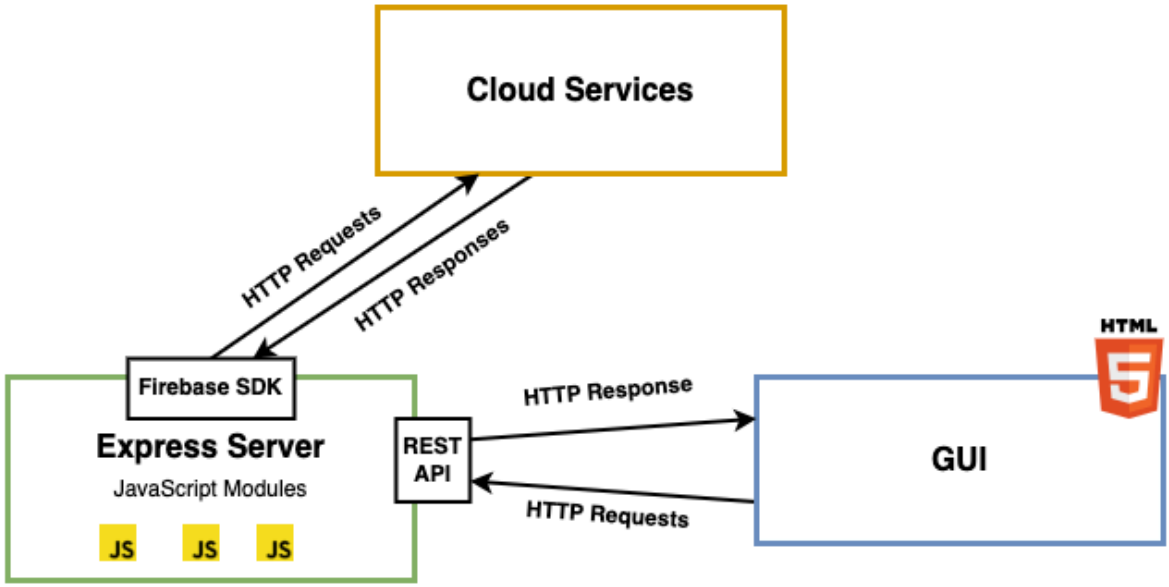
The GUI provides the following functional capabilities:

1. **User Interaction:**
    - Provide interfaces for players to interact with the game board in the web browser.
    - Enable screen interactions such as clicking to roll dice, selecting answers, and moving game pieces.
    - Implement drag-and-drop functionality for game piece movement (if applicable).

2. **Game Visualization:**
    - Render the game board and update it dynamically based on game progress.
    - Display questions and answer options in an engaging and readable format.
    - Show real-time updates of player positions, scores, and game status.
3. **Input Handling:**
    - Capture and validate user inputs for answers, dice rolls, and other game actions.
    - Provide immediate visual feedback for user actions.
4. **Communication with Server:**
    - Send requests to the server to update relevant game information.
    - Process and display server responses, updating the game state accordingly.
5. **Responsive Design:**
    - Adapt the interface for different screen sizes and orientations.
    - Ensure consistent functionality across various devices and browsers.
6. **Accessibility:**
    - Implement keyboard navigation for all game functions.
    - Provide screen reader support and ARIA labels for accessibility.
7. **Audio-Visual Enhancement:**
    - Play sound effects and background music to enhance the gaming experience.
    - Implement animations for dice rolls, player movements, and score updates.
8. **Error Handling and Feedback:**
    - Display error messages and notifications in a user-friendly manner.
    - Provide loading indicators for operations that require server communication.

Interfaces

This section describes the interfaces of each subsystem.



Server

- The Express server utilizes a REST API, enabling web pages to communicate effectively with the server through standard HTTP requests.
- Within the server, all code files are written in modular JavaScript, using import/export statements for organization.
- This modular approach facilitates discrete and manageable operations.
- WebSocket connections are implemented for real-time updates between server and clients.

Database

- Firebase services are accessed via the Firebase SDK, which standardizes the interface for database operations.
- The SDK supports operations like creating, reading, updating, and deleting data.
- For cases not covered by the SDK, HTTP requests can be made directly to the Firebase API.

GUI

- The GUI leverages HTML5 to structure web pages viewable in browsers.
- CSS3 is used for styling and responsive design.
- JavaScript handles client-side logic and user interactions.
- The GUI sends HTTP requests to the Express server's REST endpoints for game-related actions.
- WebSocket connections receive real-time updates from the server.

Functional Requirements

Please see the Use Case Document section.
Functional requirements that are not amenable to use cases are covered in the Supplemental Specification document section.

Non-Functional Requirements

Please see the Supplemental Specification document section.

Implementation Constraints

Please see the Supplemental Specification document section.

GUI Paper Prototype

Please see the GUI Paper Prototype document section. We used draw.io instead of physical paper for ease of editing.

Caffeine Coders SRS
Use Case Document

Table of Contents:

Actor Definitions 2
Use Case 1: Start New Game 3
Use Case 2: Move Player 3
Use Case 3: Answer Question on Non-Center Square 4
Use Case 4: Answer Question on Center Square 5
Use Case 5: Manage Questions 6
Use Case 6: Manage Categories 7
Use Case 7: Manage Accounts 8

Actor Definitions

Actor Name	Actor Definition
Player	A player is an individual who interacts directly with the game by engaging in activities such as answering questions, rolling the dice, and navigating the game menus. The player initiates actions within the game and expects feedback based on their interactions.
Teacher	A teacher is an individual who uses the game as an educational tool. They may set up games for students, monitor progress, and potentially customize question sets to align with curriculum goals.
Administrator	An administrator is responsible for managing the game system, including user accounts, question databases, and system settings. They have higher-level access to maintain and update the game.

Separately from the actors, there is also the game system, which is the software that manages the game logic, processes player inputs, and generates appropriate responses. It includes the server, database, and game engine components.

Use Case 1: Start New Game

Actor: Player

Description: The player initiates a new game session.

Preconditions:

1. There is no active game session for the player.

Main Flow:

1. The player indicates to start a new game.
2. The system prompts for the number of players, player names, and categories.
3. The player inputs required information.
4. The system validates the input.
5. The system initializes the game board and player positions.
6. The system displays the game board and indicates the first player's turn.

Postconditions:

1. A new game session is created and active.
2. The game board is displayed and ready for play.

Alternative Flow 1 - If the input is invalid at step 4:

1. The system displays an error message.
2. The system returns to step 2.

Use Case 2: Move Player

Actor: Player

Description: The player moves their game piece on the board.

Triggers:

1. It is the player's turn.

Main Flow:

1. The player rolls the die.
2. The system calculates the available moves based on dice roll.
3. The system highlights valid move options on the game board.
4. The player selects a destination space.
5. The system validates the move.

6. The system updates the player's position on the board.
7. The system transitions to use case 3 (answer question on non-center square).

Postconditions:

1. The player's position is updated on the game board.

Alternative Flow 1 - Player's new position is a "roll again" space at step 6:

1. The system transitions back to the start of the main flow.

Alternative postcondition for alternative flow 1:

1. The player's position is updated on the game board.

Use Case 3: Answer Question on Non-Center Square

Actor: Player

Description: The player answers a trivia question during their turn.

Preconditions:

1. It is the player's turn.

Triggers:

1. The player lands on a colored space.

Main Flow:

1. The system displays a question.
2. The player answers correctly.
3. The system transitions back to this player's turn for use case 2 (move player).

Postconditions:

1. The game state is updated.

Alternative Flow 1 - The player answers incorrectly at step 1:

1. The system transitions to the next player's turn for use case 2 (move player).

Alternative postcondition for alternative flow 1:

1. The game state is updated.

Alternative Flow 2 - This space is a headquarters space, causing a branch after step 2:

1. The system updates the player's score.
2. The system returns to the main flow at step 3.

Alternative postcondition for alternative flow 1:

1. The player's score has been updated.
2. The game state is updated.

Use Case 4: Answer Question on Center Square

Actor: Player

Description: The player answers a trivia question on the center square during their turn.

Trigger: A player lands on the center square.

Preconditions:

1. It is the player's turn.

Main Flow:

1. The system prompts for a category selection.
2. The player selects a category.
3. The system displays a question in that category.
4. The player answers correctly.
5. The system detects that the player has not collected all the category pieces.
6. The system transitions back to this player's turn for use case 2 (move player).

Postconditions:

1. The game state is updated.

Alternative Flow 1 - The player answers incorrectly at step 1:

1. The system transitions to the next player's turn for use case 2 (move player).

Alternative postcondition for alternative flow 1:

1. The game state is updated.

Alternative Flow 2 - The system detects that the player HAS collected all the category pieces, causing branch at step 5:

1. The system announces the winner.
2. The system displays final scores for all players.
3. The system updates player statistics and leaderboards.
4. The system prompts players to return to the main menu.
5. The player indicates to return to the main menu.
6. The system returns to the main menu

Alternative postconditions for alternative flow 2:

1. The game session is marked as completed.
2. Player statistics are updated in the database.
3. Players are returned to the home screen.

Use Case 5: Manage Questions

Actor: Teacher

Description: The teacher wishes to manage questions (add, modify, or delete).

Preconditions:

1. The database has at least one existing trivia question.

Main Flow:

1. The teacher selects the option to manage questions.
2. The system requests authentication.
3. The teacher enters their username and password.
4. The system displays the question manager system UI.
5. The teacher selects the existing question to modify/preview.
6. The system displays the question add/modify UI.
7. The teacher changes the features of the question (e.g. text, image, audio, category).
8. The system writes the new features of the account accordingly.
9. The teacher selects “exit preview”.
10. The system returns the teacher to the question manager system UI.
11. The teacher selects the option to return to the home screen.
12. The system returns the teacher to the home screen.

Postconditions:

1. Existing question(s) are modified.

Alternative Flow 1 - Teacher indicates to add a new question at step 5.

1. The system displays the question add/modify UI.
2. The teacher populates the features of the question (e.g. text, image, audio, category).

3. The system writes the new features of the question accordingly.
4. The teacher selects “exit preview”.
5. The system returns the teacher to the question manager system UI.

Alternative postconditions for alternative flow 1:

1. A new question is added.

Alternative Flow 2 - Teacher indicates to delete the selected question at step 6.

1. The system deletes the selected question.

Alternative postconditions for alternative flow 2:

1. The selected question is deleted.

Use Case 6: Manage Categories

Actor: Teacher

Description: The teacher wishes to manage categories (add, modify, or delete).

Preconditions:

1. The database has at least one existing category.

Main Flow:

1. The teacher selects the option to manage categories.
2. The system requests authentication.
3. The teacher enters their username and password.
4. The system displays the category manager system UI.
5. The teacher selects the existing category to modify.
6. The teacher changes the features of the account (e.g. category name).
7. The system writes the new features of the account accordingly.
8. The teacher selects the option to return to the home screen.
9. The system returns the teacher to the home screen.

Postconditions:

1. An existing category is modified.

Alternative Flow 1 - Teacher indicates to add a new category at step 5.

1. The teacher populates the features of the category (e.g. category name).

2. The system writes the new features of the category accordingly.

Alternative postconditions for alternative flow 1:

1. A new category is added.

Alternative Flow 2 - Teacher indicates to delete the selected category at step 5.

1. The system deletes the selected category.

Alternative postconditions for alternative flow 2:

1. The selected category is deleted.

Use Case 7: Manage Accounts

Actor: Teacher

Description: The teacher wishes to manage accounts (add, modify, or delete).

Preconditions:

1. The database has at least one existing teacher account.

Main Flow:

1. The teacher selects the option to manage accounts.
2. The system requests authentication.
3. The teacher enters their username and password.
4. The system displays the account manager system UI.
5. The teacher selects the existing account to modify.
6. The teacher changes the features of the account (e.g. password, access level).
7. The system writes the new features of the account accordingly.
8. The teacher selects the option to return to the home screen.
9. The system returns the teacher to the home screen.

Postconditions:

1. An existing account is modified.

Alternative Flow 1 - Teacher indicates to add a new account at step 5.

1. The teacher populates the features of the account (e.g. username, password, access level).
2. The system writes the new features of the account accordingly.

Alternative postconditions for alternative flow 1:

1. A new account is added.

Alternative Flow 2 - Teacher indicates to delete the selected account at step 5.

1. The system deletes the selected account.

Alternative postconditions for alternative flow 2:

The selected account is deleted.

Caffeine Coders SRS

Supplementary Specification

This document describes all non-functional requirements, design/implementation constraints, and any functional requirements that are not amenable to use cases.

The following figure is referenced in some requirements related to spatial location.

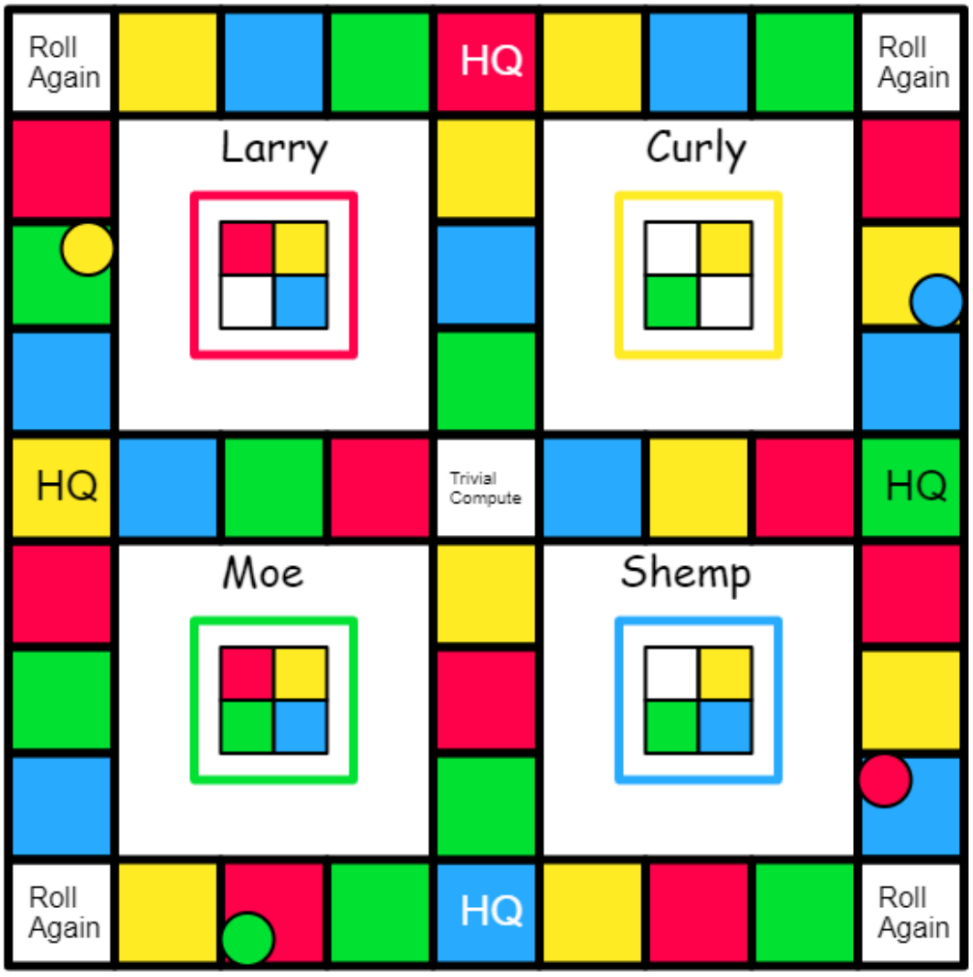


Figure 1. Layout of the game board

Non-Functional Requirements

1. The game shall support 1-4 players.
2. The game shall be able to handle an increase in the number of players without a decrease in performance.
3. The 4 score displays shall be located relative to the board layout as shown in Figure 1.
4. The game shall indicate which player's turn is active.
5. The user interface shall be intuitive and easy to navigate.
6. The game shall load within 10 seconds of launching the application.
7. The game shall maintain a consistent visual design across all platforms.
8. The game shall provide clear and prompt feedback to users when completing trivia.
9. The game shall support English.
10. The source code shall be well documented to support maintenance.
11. The game shall load video based questions within 10 seconds.

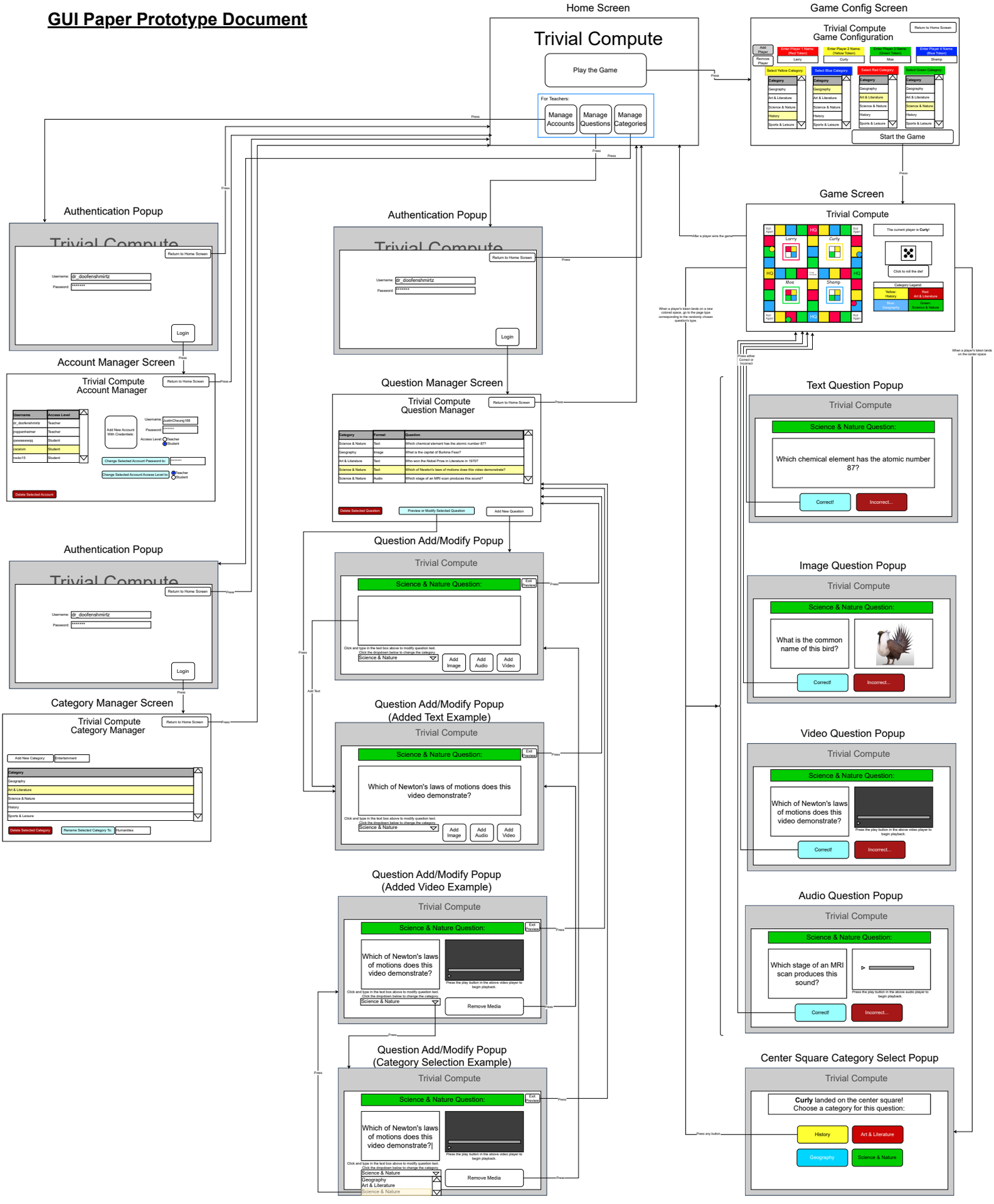
Design/Implementation Constraints

1. The game shall be developed using the specific frontend and backend framework as documented in the Software Architecture document.
2. The game shall be developed utilizing the current skill-set of the development team to avoid extensive training.
3. The project shall use open-source or free tools to minimize costs.
4. The software development shall follow the staged delivery methodology as specified in the project plan.
5. The game shall adhere to the proper rules of Trivia Pursuit.
6. All third-party libraries and frameworks used in the project shall be properly licensed and attributed.
7. Images and media files are size-limited to ensure proper loading speeds.

Functional Requirements (Non-Use Case)

1. The GUI shall support keyboard navigation for essential game functions.
2. The game shall log session data and support multiple sessions concurrently.
3. The game shall track and monitor user achievements.
4. The game shall display user-friendly error messages.
5. The game shall provide visual feedback for user-inputs.
6. The GUI shall clearly indicate the player’s turn.
7. The game shall handle user login and registration.
8. The game shall provide summarized user statistics and engagement for administrators.

GUI Paper Prototype Document



Lessons Learned

For some of the paragraphs, we used ChatGPT AI to assist with proofreading and sentence enhancement. While it provides additional information and focuses on organizing the content, it is not without flaws and requires vigilant supervision.

Credits List

Casey - Software architecture, Information and functional domain, and interfaces.

Calvin - Supplemental Specification and some work on use-cases.

Ting-Wei Wang - Use case, review and revise final document.

Justin - GUI paper prototype, some work on use cases, review and revise.